C-HTTP -- The Development of a Secure, Closed HTTP-based Network on the Internet

Takahiro Kiuchi
Department of Epidemiology and Biostatistics
Faculty of Medicine, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

Shigekoto Kaihara
Hospital Computer Center
University of Tokyo Hospital
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

Abstract

We have designed "C-HTTP" which provides secure HTTP communication mechanisms within a closed group of institutions on the Internet, where each member is protected by its own firewall. C-HTTP-based communications are made possible by the following three components: a client-side proxy, a server-side proxy and a C-HTTP name server. A client-side proxy and server-side proxy communicate with each other using a secure, encrypted protocol while communications between a user agent and client-side proxy or an origin server and server-side proxy are performed using current HTTP/1.0. In a C-HTTP-based network, instead of DNS, a C-HTTP-based secure, encrypted name and certification service is used. The aim of C-HTTP is to assure institutional level security and is different in scope from other secure HTTP protocols currently proposed which are oriented toward secure end-to-end HTTP communications in which security protection is dependent on each end-user.

1. Introduction

In the medical community, there is a strong need for closed networks among hospitals and related institutions, such as coordinating centers for clinical trials or clinical laboratories. Secure transfer of patient information for clinical use is obviously essential. In addition, some medical information has to be shared among some hospitals, but it should not be made available to other sites. This includes, for example, information concerning multi-institutional clinical trials and documents for case conferences although patients' names are usually not specified in such information. In this paper, we discuss the design and implementation of a closed HTTP (Hypertext Transfer Protocol)-based network (C-HTTP) which can be built on the Internet.

2. Design and specification of C-HTTP

2.1 Overview

C-HTTP is assumed to be used in a closed group of institutions on the Internet, in which each member is protected by its own firewall. C-HTTP-based communication is made possible with the following three components: 1) a client-side proxy on the firewall of one institution, 2) a server-side proxy on the firewall of another institution and 3) a C-HTTP name server, which manages a given C-HTTP-based network and the information for its all proxies. A client-side proxy and server-side proxy communicate with each other using a secure, encrypted protocol (C-HTTP). Communications between two kinds of proxies and HTTP/1.0 compatible servers/user agents within the firewalls are performed based on HTTP/1.0 with current C-HTTP implementation under way[1]. The DNS name service is not used for hostname resolution as the original secure name service, including certification, is used for the C-HTTP-based network. A summary of the protocol specification is described in the Appendices.

2.2 Security technology and key information

In C-HTTP, five kinds of security technologies are used. They are: 1) asymmetric key encryption for the secure exchange of data encryption keys between two types of proxies and host information between a proxy and C-HTTP name server, 2) symmetric key encryption for the encryption of C-HTTP encrypted headers and HTTP/1.0 requests, 3) electronic signature for the request/response authentication, 4) a one-way hash function for checking data tampering and 5) random key generation technology. In the C-HTTP name service, symmetric encryption is not used because the amount of information transferred is small.

Each client-side or server-side proxy has its own private and public asymmetric keys and the C-HTTP name server's public key. Proxies do not exchange their public keys with each other directly. Instead, the C-HTTP name server provides both client-side and server-side proxies with each peer's public key. In addition, Nonce values for both request and response are also generated and provided by the C-HTTP name server, which will be specified as initial values in Request-Nonce and Response-Nonce headers contained in the first C-HTTP request dispatched by a client-side proxy and in the first C-HTTP response dispatched by a server-side proxy, respectively. The C-HTTP name server manages its own private and public asymmetric keys and the public keys of all proxies which participate in the closed network. Two data encryption keys (symmetric keys) for requests and responses respectively are generated randomly during each C-HTTP session.

An origin server which is compatible with HTTP/1.0 is responsible for user authentication if necessary. It uses the built-in HTTP/1.0 authentication mechanism. Information concerning a user's ID, password and security realm (HTTP/1.0) are encrypted by proxies and are transferred only in encrypted form through the Internet. Replay attacks are blocked by checking values of the Request-Nonce header field of each request.

When a given institution wants to participate in a closed network, it must 1) install a client-side and/or server-side proxy on its firewall, 2) register an IP address (for a server-side proxy, a port number should also be registered) and hostname (which does not have to be the same as its DNS name) for a firewall gateway, 3) give the proxy's public key to the C-HTTP name server, and 4) obtain the C-HTTP name server's public key. In the present C-HTTP specification, there is only one name server in a given C-HTTP network, although one can define any possible combination of closed subnetworks within the network.

2.3 C-HTTP-based communication

C-HTTP-based communication is summarized as follows:

1) Connection of a client to a client-side proxy

A client-side proxy behaves as an HTTP/1.0 compatible proxy, and it should be specified as a proxy server for external (outside the firewall) access in each user agent within the firewall. In C-HTTP, as different from ordinary HTTP, a session (virtual C-HTTP connection) is established between a client-side proxy and server-side proxy and, thus, it is not stateless. The session is finished when the client accesses another C-HTTP server or an ordinary WWW server or when the client-side or server-side proxy times out. The following ad-hoc mechanism is employed to define a session in stateless HTTP/1.0-based communication between a client-side proxy and user agent. Suppose that the HTML specified in Figure (a) is retrieved and sent to a client-side proxy after a C-HTTP session is established. In the client-side-proxy, the HTML document is rewritten as specified in Figure (b) and forwarded to a user agent. When one of these resource names with a connection ID, for example, "http://server.in.current.connection/sample.html=@=6zdDfl dfcZLj8V!i" in Figure (b), is selected and requested by an end-user, the client-side proxy takes off the connection ID and forwards the stripped, the original resource name to the server in its request as described in Figure (c). When the connection ID is not found in the current connection table in the client-side-proxy, the current connection is disconnected. Thus a new connection is established if the host is in the closed network and an ordinary HTTP/1.0 request is dispatched otherwise.

2) Lookup of server-side proxy information (Appendix 3. a,b)

A client-side proxy asks the C-HTTP name server whether it can communicate with the host specified in a given URL. If the name server confirms that the query is legitimate, it examines whether the requested server-side proxy is registered in the closed network and is permitted to accept the connection from the client-side proxy. If the connection is permitted, the C-HTTP name server sends the IP address and public key of the server-side proxy and both request and response Nonce values. If it is not permitted, it sends a status code which indicates an error. If a client-side proxy receives an error status, then it performs DNS lookup, behaving like an ordinary HTTP/1.0 proxy.

Both the request to and response from the C-HTTP name server are encrypted and certified, using asymmetric key encryption and digital signature technology.

3) Request for connection to the server-side proxy (Appendix 3. c)

When the C-HTTP name server confirms that the specified server-side proxy is an appropriate closed network member, a client-side proxy sends a request for connection to the server-side proxy, which is encrypted using the server-side proxy's public key and contains the client-side proxy's IP address, hostname, request Nonce value and symmetric data exchange key for request encryption.

4) Lookup of client-side proxy information (Appendix 3. d,e)

When a server-side proxy accepts a request for connection from a client-side proxy, it asks the C-HTTP **Figure. Conversion of stateless HTTP to stateful C-HTTP**

a. The HTML document sent from a origin server to a client-side proxy

```
<TITLE>SAMPLE</TITLE>
<BODY>
<A HREF =
"http://server.in.current.connection/sample.html">
Please click here.</A>
<A HREF =
"http://another.server.in.closed.network/">
Another server.</A>
</BODY>
```

b. The HTML document rewritten and forwarded to a use agent by the client-side proxy. The string, "6zdDfldfcZLj8V!i", attached to the end of the URLs is a connection ID

```
<TITLE>SAMPLE</TITLE>
<BODY>
<A HREF =
"http://server.in.current.connection/sample.html=@=6zdDfldfcZ
Lj8V!i">
Please click here.</A>
<A HREF =
"http://another.server.in.closed.network/=@=6zdDfldfcZLj8V!i"
>
Another server.</A>
</BODY>
```

c. HTTP/1.0 request from the user agent (1) and HTTP/1.0 request encrypted and wrapped in C-HTTP request dispatched by the client-side proxy (2)

```
(1)
GET "http://server.in.current.connection/
sample.html=@=6zdDfldfcZLj8V!i" HTTP/1.0<CR><LF>

(2)
GET "http://server.in.current.connection/
sample.html"
HTTP/1.0<CR><LF>
```

name server whether the client-side proxy is an appropriate member of the closed network. If the name server confirms that the query is legitimate, it then examines whether the client-side proxy is permitted to access to the server-side proxy. If access is permitted, the C-HTTP name server sends the IP address and public key of the client-side proxy and both request and response Nonce values, which are the same as those sent to the client-side proxy. The C-HTTP name server keeps both of the Nonce values for thirty seconds. If not, it sends a status code which indicates an error and the server-side proxy refuses the connection from the client-side proxy.

5) Connection establishment (Fig. 2f)

When the sever-side proxy obtains the client-side proxy's IP address, hostname and public key, it authenticates the client-side proxy, checks the integrity of the request and the request Nonce value and generates both a connection ID derived from the server-side proxy's name, date and random numbers (32 bits) using MD5, and also a second symmetric data exchange key for response encryption, which are sent to the client-side proxy. When the client-side proxy accepts and checks them, the connection is established.

6) Sending C-HTTP requests to the server-side proxy (Fig. 2g)

Once the connection is established, a client-side proxy forwards HTTP/1.0 requests from the user agent in encrypted form using C-HTTP format.

7) Forwarding requests to an origin server

Using HTTP/1.0, a server-side proxy communicates with an origin server inside the firewall. From the view of the user agent or client-side proxy, all resources appear to be located in a server-side proxy on the firewall. In reality, however, the server-side proxy forwards requests to the origin server. It is possible to map any of the virtual directories on the server-side proxy to any of the directories in one or more origin servers inside the firewall.

8) Origin server responses to the user agent through the server-side and client-side proxies (Fig. 2h)

An HTTP/1.0 response sent from the origin server to the server-side proxy is encrypted in C-HTTP format by the server-side proxy, and is forwarded to the client-side proxy. Then, in the client-side proxy, the C-HTTP response is decrypted and the HTTP/1.0 response extracted. If the transferred object is in HTML format, the connection ID is attached to the anchor URLs contained in the document. The resulting HTTP/1.0 response is sent to the user agent.

9) Request for closing the connection (Appendix 3. i,j)

A client-side proxy can send a request for closing the connection. The server-side proxy returns a status which indicates the connection is closed. On the other hand, if the server-side proxy detects that a given connection times out, it deletes the connection ID from the connection list, informing the client-side proxy that the connection is closed when an error status is returned in response to the request.

3. Trial implementation

Trial implementation is under way using 1) RSA as the asymmetric key encryption method (OSISEC RSA library)[2], 2) DES as the symmetric key encryption method (GNU DES library)[3], 3) RSA as the electronic signature method (OSISEC RSA library) and 4) a one-way hash function based on MD5[4]). As for random key generation, programs included in the OSISEC RSA library and GNU DES library are used for RSA asymmetric keys and DES symmetric keys, respectively.

In the implementation, we employed the following methods to enhance security.

1) Key protection

In C-HTTP, keys are stored only on the firewall of a given institution. C-HTTP proxy software is provided as source code, and the keys are designed not to be stored in a separate "key file." A key generation program generates a C program file, which contains key information for proxies. It is more difficult to steal keys using this method than if they were stored in a separate file.

2) No simultaneous data transfer to both sides

Only after receiving all the data transferred from one side, does a proxy server begin to forward it to the other side, except for image and sound data. In this method, the performance of data transfer is not good, however, the data transfer is separated between the internal and external sides. For the secure implementation of this feature, the size of HTML documents and object bodies should be limited and checked by each proxy. We plan to implement routines which check the contents of object bodies (especially concerning form data used in POST method) in the future.

3) Closure of TCP connection after each transaction

C-HTTP itself is stateful, but the TCP connection is closed after each transaction (request and response pair) in order to reduce the possibility of it being intercepted by attackers.

4. Discussion

4.1 Why HTTP?

It is possible to develop a secure application level protocol available only to a closed group in the Internet, making use of cipher technology. The reasons we chose HTTP as the communication protocol for a closed network are as follows:

1) Flexibility of HTTP

Different application level protocols have been developed for individual network services, such as FTP, SMTP, NNTP or GOPHER[5],[6],[7],[8]. HTTP has the flexibility to be able to provide services similar to those which have been provided by these protocols . For example, file transfer by FTP is accomplished by the object transfer mechanism of HTTP and, from a functional viewpoint, the Gopher protocol can be considered a subset of HTTP. Internet news and electronic mail services are available with an HTTP-based graphical user interface via gateways for protocol conversions[9]. Electronic mail services within a given group of institutions can be also developed using HTTP and CGI (Common Gateway Interface)[10].

2) Hypertext-based user-friendly graphical interface

Using HTTP and the Hypertext Markup Language (HTML), distributed multimedia information systems with user-friendly graphical interfaces based on hypertext can be easily developed[11].

3) User agents and servers available on almost all platforms

HTTP has now gained widespread popularity and various kinds of user agents and servers are available on almost all platforms. Even if new protocols for closed networks are developed which are superior in function or flexibility, new clients and servers have to be developed for compatibility, which is costly and an obstacle to their universal acceptance.

4.2 Proxy-proxy vs. end-to-end secure HTTP-based information exchange

As for hospitals, from which the Internet is available, in-hospital networks are usually protected using a dual home gateway and packet filter (firewall) and the Internet can only be accessed through proxies on the firewalls. The role of proxies in HTTP communication has been considered as important in communicating over firewalls and transferring information efficiently by caching. Other secure HTTP protocols are designed to be implemented in origin servers and user agents in order to assure "end-to-end" security protection[12-15]. Our approach is aimed at assuring proxy-proxy security and is fundamentally different from theirs.

All proposals for secure HTTP communications are designed to be secure against the following attacks: 1) network tampering, 2) replay attacks and 3) middle of the man attack[12-15]. C-HTTP is also designed to be secure against these attacks and, in addition, it has the following enhancements for security protection.

1) No end-user has any chance to obtain keys for encryption or decryption.

Much cost and time are necessary to decode ciphers which have been used for a long time and are considered confidential, such as DES or RSA, so an easier and more practical way to obtain original information is not to decode them, but to "steal a key" instead. It is not realistic for hospital information managers to expect that all individual end-users, including those who connect their PCs to in-hospital LANs, manage their keys in a secure manner.

As currently proposed secure HTTP protocols aim at providing end-to-end security mechanisms, responsibility for security is attributed to each individual user. Secure transfer of data exchange keys is performed by exchanging public keys (in most cases with certificates) between both parties. In this situation, once a private key is stolen, it is possible to obtain information from WWW servers outside the hospital.

Undoubtedly, the purpose of security protection is secure commercial information services or on-line shopping services which are provided by profit-making companies for the masses. For commercial services, it is reasonable that individual users (payers) are responsible for "their own risks," but, as for patient information, it is each hospital that should be responsible for "their patients' risks." Each hospital should take measures to assure security at the institutional level.

2) Name service

As C-HTTP includes its own secure name service, which contains a certification mechanism, it is impossible to know the IP address of a server-side proxy even if its C-HTTP hostname (not necessarily the same as its DNS name) is known and vice versa. The C-HTTP name service is efficient because it can do name resolution and host certification simultaneously.

3) Difficulty in accessing from outside the closed network

It is difficult to access any servers in a closed network from outside. A cracker has to take the following steps:

a) To find the IP address and port number of a server-side proxy
b) To get the public key of the server-side proxy in order to send a valid C-HTTP request for C-HTTP connection.
c) To make a TCP connection to a target server-side proxy using a certain client-side proxy's IP address
d) To make the server-side proxy believe that request comes from a legitimate client-side proxy within the closed network. For this, it is necessary to know the private key and C-HTTP hostname of the client-side proxy.

There are other merits in favor of C-HTTP over other secure HTTP protocols, although they are not the original purposes of the development.

1) Easy installation

A C-HTTP based network is made available simply by installing proxies on the firewall and registering their information with the C-HTTP name server. Current HTTP/1.0 compatible servers and clients can be used as they are.

2) Simplicity

There are no negotiations concerning security options or type and representation of objects in C-HTTP because C-HTTP-based communication is performed only between two types of C-HTTP proxies and between a C-HTTP proxy and C-HTTP name server. They do not communicate directly with various types of user agents and servers using C-HTTP. Negotiations concerning type and representation of objects are done between an origin server and user agent, using HTTP/1.0. As for these negotiations, C-HTTP is transparent to both of them. This makes the design and implementation of C-HTTP simple.

3) Easy manipulation by end-users

End-users do not have to employ security protection procedures. They do not even have to be conscious of using C-HTTP based communications.

4.3 Disadvantages and limitations

Our proposal has some disadvantages and limitations, and it should be used where its use is appropriate and suitable, taking them into account.

The key technology used in the Internet is dedicated to assure connectivity between the huge number of computers, which may be added or removed at any time. Such connectivity is attractive to commercial companies and, in this context, it is necessary to develop technology which assures secure communications between a huge number of computers.

Our system is assumed to accommodate up to a few hundreds proxies. This number is much smaller than that needed for most commercial purposes. In addition, a new proxy should be registered manually to the centralized name server. For the management of huge number of proxies, another mechanism for proxy management is necessary.

4.4 Relations to other secure HTTP protocols

C-HTTP is not an alternative to other secure HTTP proposals, but it can co-exist with them. Although the current C-HTTP implementation assumes the use of HTTP/1.0 compatible user agents and servers, it is possible to develop C-HTTP proxies which can communicate with other secure HTTP compatible user agents and servers. If

C-HTTP is used with these protocols, which assure end-to-end or individual security, both institutional and personal level security protection can be provided. This means that even if individual security management is not sufficient, data security can be guaranteed. In this case, administrators of proxies on the firewall can not know the contents of any information exchanged.

There are commercial proxy servers which can communicate externally using secure HTTP and internally using either HTTP/1.0 or secure HTTP, such as a NetScape Proxy Server. This feature is designed to ensure that clients which do not support secure HTTP can communicate with external secure HTTP servers in a secure manner. Such a proxy server is oriented to be open to any external clients or servers, although the access restrictions and authentications can be set up in each proxy server. A set of such proxy servers can constitute a closed network if appropriate access restrictions and authentications are set up in all of the members. However, this method does not always meet our requirements as follows:

1) Centralized management of a closed network

In the Internet, there is no "central" network manager. The huge number of computers on the Internet makes it impossible. In our closed network, we adopt centralized proxies and network management. It may be troublesome for each proxy manager in a hospital to set up his or her proxy in order to make it accessible only to or from "medical sites."

2) Closed network's own name service is necessary

It is desirable for us not to make computers dealing with sensitive medical information known to the public. It is one method to enhance security. On the other hand, we have to inform medical researchers of the locations of medical information resources using publicly available media, such as medical journals or research protocols for clinical trials.

Besides the above mentioned reasons, it is desirable for us, or the medical community, to obtain license-free software with source code. It is not always necessary for a given closed group to adopt "standards." It can modify C-HTTP or develop its own new protocol, based on C-HTTP.

4.5 Privately-leased circuits plus the Internet vs. the Internet alone

The Internet is expected to become available to almost all major hospitals. Although a closed network can be constructed using a privately-leased circuit, additional investment for its construction is necessary. If a closed network can be constructed on the Internet, it would be convenient, speedy and reasonable in terms of cost. In addition, if a closed network is realized by privately-leased circuits, it is not always easy to operate several closed networks flexibly and simultaneously. Combinations of institutions may alter according to several different purposes. For example, in multi-institutional clinical trials, the hospitals which participate in each trial usually differ and the length of each trial is usually a few years.

5. Concluding remarks

Although C-HTTP is primarily developed for use in the medical field, it can be used in other areas. Using C-HTTP, a closed HTTP-based virtual network can be constructed for closed groups; for example, the headquarters and branches of a given corporation. This kind of usage may not fit with the spirit of the Internet, but if resources which might otherwise be invested in private circuits are channeled into the Internet, it will contribute to its further development.

6. References

[1] Berners-Lee T, Fielding RT, Nielsen HF. Hypertext Transfer Protocol -- HTTP/1.0. Internet Draft, 1995 (Work in progress, available on the World Wide Web as "ftp://ds.internic.net/internet-drafts/draft-ietf-http-v10-spec-00.txt")

[2] Roe M, Hardcastle-Kille S, Williams P, Kirstein P. OSISEC RSA Library, 1995 (Available on the World Wide Web as "ftp://cs.ucl.ac.uk/osisec/IC-OSISEC-V2.3.tar.des")

[3] Young E. GNU DES library version 3.00. Free Software Foundation, 1993

[4] Rivest R. The MD5 Message-Digest Algorithm. RFC 1321,1992

[5] Postel J, Reynolds J. File Transfer Protocol (FTP). RFC 959, 1985

[6] Postel JB. Simple Mail Transfer Protocol. RFC 821, 1982

[7] Kantor B, Lapsley P. Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News. RFC 977, 1986

[8] Anklesaria F, McCahill M, Lindner P, Johnson D, Torrey D, Alberti B. The Internet Gopher Protocol (a distributed document search and retrieval protocol), RFC 1436, 1993

[9] Yahoo. Computers and Internet:Internet:World Wide Web:Gateways, 1995 (Available on the World Wide Web as_@"http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/Gatways/")

[10] McCool R. The Common Gateway Interface, 1995 (Available on the World Web as "http://hoohoo.ncsa.uiuc.edu/cgi/overview.html")

[11] Raggett D. Hypertext Markup Language Specification Version 3.0. Internet Draft, 1995 (Work in progress, available on the World Wide Web as "ftp://ds.internic.net/internet-drafts/draft-ietf-html-specv3-00.txt")

[12] Rescorla E, Schiffman A. The Secure Hypertext Transfer

Protocol. Internet Draft, 1995 (Work in progress, available on the World Wide Web as "ftp://ds.internic.net/internet-drafts/draft-ietf-wts-shttp-00.txt")
[13]Hallam-Baker PM. Shen: A Security Scheme for the World Wide Web. 1995 (Available on the World Wide Web as "ftp://www.w3.org/hypertext/WWW/Shen/ref/security_spec.html")
[14] Hickman KEB, Elgamal T. The SSL Protocol. Internet Draft, 1995 (Work in progress, available on the World Wide Web as "ftp://ds.internic.net/internet-drafts/draft-hickman-netscape-ssl-01.txt")
[15] Spero S. Progress on HTTP-NG. (Available on the World Wide Web as "http://www.w3.org/hypertext/WWW/Protocols/HTTP-NG/http-ng-status.html")

Appendices.

Appendix 1. Summary of the C-HTTP specification

Note that lines with an asterisk are encrypted and that all specifications are described in the notation (augmented BNF) used in the HTTP/1.0 specification[1].

1. C-HTTP request

```
Request-Line
Plain-Header
CRLF
*General-Header
*Request-Header
*HTTP/1.0-Request
CRLF
Digital-Signature
```

1.1 Request-Line

```
Request-Line=Request-MethodSP C-HTTP-Version CRLF
Request-Method  = "CONNECT"
                | "REQUEST"
                | "CLOSE"
C-HTTP-Version  = "C-HTTP/0.7"
```

1.1.1 Request-Method
1) CONNECT
Used when a client-side proxy requests a C-HTTP connection to a server-side proxy
2) REQUEST
Used when a client-side proxy sends a C-HTTP request to a server-side proxy, to which C-HTTP connection has already been established.
3) CLOSE
Used when a client-side proxy closes an established connection to a server-side proxy

1.2 Plain-Header

```
Request-Plain-Header =  Encryption-Algorithm
                     | Encrypted-Header-Length
                     | Encrypted-Request-Length
                     | Signature-Algorithm
                     | Signature-Length
                     | Message-Digest-Algorithm
```

1) Encryption-Algorithm:
Used for specifying the encryption algorithm.
2) Encrypted-Header-Length:
Used for specifying header length.
3) Encrypted-HTTP-1.0-Length:
Used for specifying HTTP/1.0 request/response length.
4) Signature-Algorithm:
Used for specifying the signature algorithm.
5) Signature-Length:
Used for specifying signature length.
6) Message-Digest-Algorithm:
Used for specifying the message digest algorithm.

1.3 General-Header

```
Request-Header =  Client-Side-Proxy-IP
             | Client-Side-Proxy-Name
               | Server-Side-Proxy-IP
               | Server-Side-Proxy-Name
             | Server-Side-Proxy-Port
               | Connection-ID | User-Agent-IP
```

1) Client-Side-Proxy-IP:

Used for specifying the IP address of a client-side proxy.
2) Client-Side-Proxy-Name:
Used for specifying the hostname of a client-side proxy.
3) Server-Side-Proxy-IP:
Used for specifying the IP address of a server-side proxy.
4) Server-Side-Proxy-Name:
Used for specifying the hostname of a server-side proxy.
5) Server-Side-Proxy-Port: Used for specifying the port number of a server-side proxy.
7) Connection-ID:
Used for specifying the connection ID.
8) User-Agent-IP:
Used for specifying the IP address of a user agent.

1.4 Request-Header

```
Request-Header =  Request-Data-Exchange-Key
                  | Request-Nonce
```

1) Request-Data-Exchange-Key: Used for specifying a data exchange key for response encryption when the connection is established.
2) Request-Nonce: Used for specifying the Nonce value of a request.

1.5 Digital-Signature
Digital-Signature = <Message digest of the remainder of the request>

2. C-HTTP Response

C-HTTP-Version-Line
Plain-Header
CRLF
*General-Header
*Response-Header
*HTTP/1.0-RESPONSE
CRLF
Digital-Signature

2.1 C-HTTP-Version-Line
C-HTTP-Version-Line = "C-HTTP/0.7" CRLF

2.2 Plain-Header (the same as those in C-HTTP Request)

2.3 General-Header (the same as those in C-HTTP Request)

2.4 Response-Header (Note that headers used only in responses are described.)

```
Response-Header = Status
                  | Response-Data-Exchange-Key
                  | Server-Side-Proxy-Public-Key
                  | Client-Side-Proxy-Public-Key
                  | Response-Nonce
```

1) Status:
Used for specifying the server's status in response to a request.
2) Response-Data-Exchange-Key:
Used for specifying the data exchange key for response when the connection is established.
3) Server-Side-Proxy-Public-key:
Used in the C-HTTP name server's response for specifying the server-side proxy's public key.
4) Client-Side-Proxy-Public-Key:
Used in the C-HTTP name server's response for specifying the client-side proxy's public key.
5) Response-Nonce:
Used for specifying the Nonce value of a response.

2.5 Digital Signature

Digital-Signature = <Message digest of the remainder of the response>

Appendix 2. The summary of C-HTTP name service protocol.

Note that lines with an asterisk are encrypted. C-HTTP name service protocol is different from other parts of the C-HTTP in the following points:

1) Only asymmetric key encryption is used for the encryption.
2) There are no headers, which precede actual data. The order of data
transferred is fixed and cannot be changed.
3) Random bytes are inserted every fourth byte of the request and response before encryption in order to avoid the same encrypted requests or responses being repeated. This is useful to make it impossible to predict the server-side-proxy which will be connected after the name request.

1. Notations

In the C-HTTP name service protocol, the notations below are used for the description of the protocol.

1.1 Plain information

1) C-HTTP-NAME-SERVICE-VERSION:
Version of C-HTTP name service protocol
2) ENCRYPTION-ALGORITHM:
Asymmetric encryption algorithm used
3) ENCRYPTED-PART-LENGTH:
Length of encrypted part
3) SIGNATURE-ALGORITHM:
Electronic signature algorithm used
4) SIGNATURE-LENGTH:
Length of electronic signature
5) MESSAGE-DIGEST-ALGORITHM:
Message digest algorithm

1.2 Encrypted information

1) REQUEST-TYPE:
Type of C-HTTP name request. Options available now are "SERVER" for a server-side proxy information and "CLIENT" for a client-side proxy information.
2) C-HTTP-NAME-SERVICE-STATUS:
C-HTTP name server's status returned to proxies. Possible status is "OK" or "Disallowed."
3) CLIENT-SIDE-PROXY-IP:
IP address a client-side proxy
4) CLIENT-SIDE-PROXY-NAME:
A name of a client-side proxy
5) CLIENT-SIDE-PROXY-PUBLIC-KEY:
A public key of a client-side proxy
6) SERVER-SIDE-PROXY-IP:
IP address of a server-side proxy
7) SERVER-SIDE-PROXY-NAME:
A name of a server-side proxy
8) SERVER-SIDE-PROXY-PORT:
A port number of server-side proxy
9) SERVER-SIDE-PROXY-PUBLIC-KEY:
A public key of a server-side proxy
10) USER-AGENT-IP:
IP address of a user agent
11) REQUEST-NONCE:
A request-Nonce value provided with both type of proxies
12) RESPONSE-NONCE:
A response-Nonce value provided with a server-side proxy

1.3 Signature

DIGITAL-SIGNATURE: Digital signature

2. Communications between a client-side proxy and C-HTTP name server

2.1 C-HTTP name service request

```
C-HTTP-NAME-SERVICE-VERSION<CR><LF>
ENCRYPTION-ALGORITHM<CR><LF>
ENCRYPTED-PART-LENGTH<CR><LF>
SIGNATURE-ALGORITHM<CR><LF>
SIGNATURE-LENGTH<CR><LF>
MESSAGE-DIGEST-ALGORITHM<CR><LF>
<CR><LF>
*REQUEST-TYPE<CR><LF>
*CLIENT-SIDE-PROXY-IP<CR><LF>
*USER-AGENT-IP<CR><LF>
*SERVER-SIDE-PROXY-NAME<CR><LF>
*SERVER-SIDE-PROXY-PORT<CR><LF>
<CR><LF>
*DIGITAL-SIGNATURE
```

2.2 C-HTTP name service response

```
ENCRYPTION-ALGORITHM<CR><LF>
ENCRYPTED-PART-LENGTH<CR><LF>
SIGNATURE-ALGORITHM<CR><LF>
SIGNATURE-LENGTH<CR><LF>
MESSAGE-DIGEST-ALGORITHM<CR><LF>
<CR><LF>
*C-HTTP-NAME-SERVICE-STATUS<CR><LF>
*CLIENT-SIDE-PROXY-IP<CR><LF>
```

```
*USER-AGENT-IP<CR><LF>
*SERVER-SIDE-PROXY-IP<CR><LF>
*SERVER-SIDE-PROXY-PORT<CR><LF>
*SERVER-SIDE-PROXY-PUBLIC-KEY<CR><LF>
*REQUEST-NONCE<CR><LF>
*RESPONSE-NONCE<CR><LF>
<CR><LF>
*DIGITAL-SIGNATURE
```

3. Communications between a sever-side proxy and C-HTTP name server

3.1 C-HTTP name request

```
C-HTTP-NAME-SERVICE-VERSION<CR><LF>
ENCRYPTION-ALGORITHM<CR><LF>
ENCRYPTED-PART-LENGTH<CR><LF>
SIGNATURE-ALGORITHM<CR><LF>
SIGNATURE-LENGTH<CR><LF>
MESSAGE-DIGEST-ALGORITHM<CR><LF>
<CR><LF>
*REQUEST-TYPE<CR><LF>
*SERVER-SIDE-PROXY-IP<CR><LF>
*SERVER-SIDE-PROXY-PORT<CR><LF>
*CLIENT-SIDE-PROXY-NAME<CR><LF>
*USER-AGENT-IP<CR><LF>
<CR><LF>
*DIGITAL-SIGNATURE
```

3.2 C-HTTP name service response

```
ENCRYPTION-ALGORITHM<CR><LF>
ENCRYPTED-PART-LENGTH<CR><LF>
SIGNATURE-ALGORITHM<CR><LF>
SIGNATURE-LENGTH<CR><LF>
MESSAGE-DIGEST-ALGORITHM<CR><LF>
<CR><LF>
*C-HTTP-NAME-SERVICE-STATUS<CR><LF>
*SERVER-SIDE-PROXY-IP<CR><LF>
*SERVER-SIDE-PROXY-PORT<CR><LF>
*CLIENT-SIDE-PROXY-IP<CR><LF>
*CLIENT-SIDE-PROXY-PUBLIC-KEY<CR><LF>
*USER-AGENT-IP<CR><LF>
*REQUEST-NONCE<CR><LF>
*RESPONSE-NONCE<CR><LF>
<CR><LF>
*DIGITAL-SIGNATURE
```

Appendix 3. Examples of C-HTTP communication (a-h)

Note that lines with an asterisk are encrypted. Components of C-HTTP-based communication are as follows:

1) Client-side proxy
   hostname: University.of.Tokyo.Branch.Hospital
   IP address: 130.69.111.111
2) server-side proxy
   hostname: Coordinating.Center.CSCRG
   IP address: 130.69.222.222
   port number: 8080
3) C-HTTP name server:
   Name.Server.CSCRG
   IP address: 130.69.222.111
4) User agent:
   IP address: 192.168.123.123

a. Lookup of server-side proxy information (C-HTTP name service protocol)

```
C-HTTPNS/0.1<CR><LF>
RSA<CR><LF>
74<CR><LF>
RSA<CR><LF>
32<CR><LF>
MD5<CR><LF>
<CR><LF>
*SERVER<CR><LF>
*130.69.111.111<CR><LF>
*192.168.123.123<CR><LF>
*Coordinating.Center.CSCRG<CR><LF>
*8080<CR><LF>
<CR><LF>
*827ae79ba214769ea2998249bdb9aa97
```

b. Response from the C-HTTP name server, indicating that the connection is permitted (C-HTTP name service protocol)

```
RSA<CR><LF>
203<CR><LF>
RSA<CR><LF>
32<CR><LF>
MD5<CR><LF>
<CR><LF>
*OK<CR><LF>
*130.69.111.111<CR><LF>
```

```
*192.168.123.123<CR><LF>
*130.69.222.222<CR><LF>
*8080<CR><LF>
*effe0d7f480dad7cbbe9d0c309b2f04c89fe5e8e9f7bfc1854b
62f6b4bafa981c1a64e19fd6c702eec376f9dea4f5422e851bb
1770ce600d246637459ab757b<CR><LF>
*8abd853f<CR><LF>
*ef23dc99<CR><LF>
*<CR><LF>
<CR><LF>
*51a2f15a51a7f64a5ada6ae40bc529ba
```

c. Request for connection to the server-side proxy

```
CONNECT C-HTTP/0.7<CR><LF>
Encryption-Algorithm: RSA<CR><LF>
Encrypted-Header-Length: 298<CR><LF>
Signature-Algorithm: RSA<CR><LF>
Signature-Length: 32<CR><LF>
Message-Digest-Algorithm: MD5<CR><LF>
<CR><LF>
*Client-Side-Proxy-IP: 130.69.109.111.111<CR><LF>
*Client-Side-Proxy-Name:
University.of.Tokyo.Branch.Hospital<CR><LF>
*User-Agent-IP: 192.168.123.123<CR><LF>
*Server-Side-Proxy-IP: 130.69.222.222 <CR><LF>
*Server-Side-Proxy-Name:
Coordinating.Center.CSCRG<CR><LF>
*Server-Side-Proxy-Port: 8080<CR><LF>
*Data-Exchange-Key-Request: #8Jk=d.n,&1i^s<CR><LF>
*Request-Nonce: 8abd853f<CR><LF>
<CR><LF>
*4d3b8ea8060046a38d1a3aab34ce2cb8
```

d. Lookup of client-side proxy information (C-HTTP name service protocol)

```
C-HTTPNS/0.1<CR><LF>
RSA<CR><LF>
83<CR><LF>
RSA<CR><LF>
32<CR><LF>
MD5<CR><LF>
<CR><LF>
*CLIENT<CR><LF>
*130.69.222.222<CR><LF>
*8080<CR><LF>
*University.of.Tokyo.Branch.Hospital<CR><LF>
*192.168.123.123<CR><LF>
<CR><LF>
*d6cdf3fc703270d62ad757853a4efdd4
```

e. Response from the C-HTTP name server, indicating that the connection is permitted (C-HTTP name service protocol)

```
RSA<CR><LF>
202<CR><LF>
RSA<CR><LF>
32<CR><LF>
MD5<CR><LF>
<CR><LF>
*OK<CR><LF>
*130.69.222.222<CR><LF>
*8080<CR><LF>
*130.69.111.111<CR><LF>
*c02c5351910eca0d1fbde540a0fc8e6b9ef7a83582d286b049
e690d67e95b74574ff88207d6630a850b7789625c299e47317
290a3f082b2f96037c60ee11f7cb<CR><LF>
*192.168.123.123<CR><LF>
*8abd853f<CR><LF>
*ef23dc99<CR><LF>
<CR><LF>
*2bcd8b04e019a8510bc56902bede92bb
```

f. Response from the server-side proxy, indicating that the connection is established

```
C-HTTP/0.7<CR><LF>
Encryption-Algorithm: RSA<CR><LF>
Encrypted-Header-Length: 341<CR><LF>
Signature-Algorithm: RSA<CR><LF>
Signature-Length: 32<CR><LF>
Message-Digest-Algorithm: MD5<CR><LF>
<CR><LF>
*Status: 200 OK<CR><LF>
*Server-Side-Proxy-IP: 130.69.222.222<CR><LF>
*Server-Side-Proxy-Name:
Coordinating.Center.CSCRG<CR><LF>
*Server-Side-Proxy-Port: 8080<CR><LF>
*Client-Side-Proxy-IP: 130.69.111.111<CR><LF>
*Client-Side-Proxy-Name:
University.of.Tokyo.Branch.Hospital<CR><LF>
*User-Agent-IP: 192.168.123.123<CR><LF>
*Connection-ID: 6zdDfldfcZLj8V!i<CR><LF>
*Response-Nonce: ef23dc99<CR><LF>
*Response-Data-Exchange-Key: a-3f(*d.bfs,<CR><LF>
<CR><LF>
*36e2bfc5022208ca8c20307f60d15e2e
```

g. Sending C-HTTP requests to the server-side proxy

```
REQUEST C-HTTP/0.7<CR><LF>
Encryption-Algorithm:DES-CBC, f80a9cb01a93a8df
<CR><LF>
Encrypted-Header-Length: 349<CR><LF>
Encrypted-HTTP-1.0-Length: 31<CR><LF>
Signature-Algorithm: RSA<CR><LF>
Signature-Length: 32<CR><LF>
Message-Digest-Algorithm: MD5<CR><LF>
<CR><LF>
*Server-Side-Proxy-IP: 130.69.222.222<CR><LF>
*Server-Side-Proxy-Name:
Coordinating.Center.CSCRG<CR><LF>
*Server-Side-Proxy-Port: 8080<CR><LF>
*Client-Side-Proxy-IP: 130.69.111.111<CR><LF>
*Client-Side-Proxy-Name:
University.of.Tokyo.Branch.Hospital<CR><LF>
*User-Agent-IP: 192.168.123.123<CR><LF>
*Connection-ID: 6zdDfldfcZLj8V!i<CR><LF>
*Request-Nonce: 8abd8540<CR><LF>
*<HTTP/1.0 Request>
<CR><LF>
*2ec7d37d0ac2c15ddc455c45913affe6
```

h. Response from the server-side proxy, indicating that the request is successful

```
C-HTTP/0.7<CR><LF>
Encryption-Algorithm: DES-CBC,
ae428ffea1248ac3<CR><LF>
Encrypted-Header-Length: 349<CR><LF>
Encrypted-HTTP-1.0-Length: 4121<CR><LF>
Signature-Algorithm: RSA
Signature-Length: 32<CR><LF>
Message-Digest-Algorithm: MD5<CR><LF>
<CR><LF>
*Status: 200 OK<CR><LF>
*Server-Side-Proxy-IP: 130.69.222.222<CR><LF>
*Server-Side-Proxy-Name:
Coordinating.Center.CSCRG<CR><LF>
*Server-Side-Proxy-Port: 8080<CR><LF>
*Client-Side-Proxy-IP: 130.69.111.111<CR><LF>
*Client-Side-Proxy-Name:
University.of.Tokyo.Branch.Hospital<CR><LF>
*User-Agent-IP: 192.168.123.123<CR><LF>
*Connection-ID: 6zdDfldfcZLj8V!i<CR><LF>
*Response-Nonce: ef23dc9a<CR><LF>
*<HTTP/1.0 Response>
<CR><LF>
*5471c62cd0267fd02c4789fe679e5eb9
```

i. Request for closing the connection

```
CLOSE<SP> C-HTTP/0.7<CR><LF>
Encryption-Algorithm: DES-CBC,
acd42bef76gf3b98<CR><LF>
Encrypted-Header-Length: 349<CR><LF>
Signature-Algorithm: RSA<CR><LF>
Signature-Length: 32<CR><LF>
Message-Digest-Algorithm: MD5<CR><LF>
<CR><LF>
*Client-Side-Proxy-IP: 130.69.111.111<CR><LF>
*Client-Side-Proxy-Name:
University.of.Tokyo.Branch.Hospital<CR><LF>
*User-Agent-IP: 192.168.123.123<CR><LF>
*Server-Side-Proxy-IP: 130.69.222.222<CR><LF>
*Server-Side-Proxy-Name:
Coordinating.Center.CSCRG<CR><LF>
*Server-Side-Proxy-Port: 8080<CR><LF>
*Connection-ID: 6zdDfldfcZLj8V!i<CR><LF>
*Request-Nonce: 8abd8541<CR><LF>
<CR><LF>
*c37d7b17f13efce24e7a9ae6859d0293
```

j. Response from the server-side proxy, indicating that the connection closed

```
C-HTTP/0.7<CR><LF>
Encryption-Algorithm: DES-CBC,
178dacfff4gf3b3a<CR><LF>
Encrypted-Header-Length: 349<CR><LF>
Signature-Algorithm: RSA<CR><LF>
Signature-Length: 32<CR><LF>
Message-Digest-Algorithm: MD5<CR><LF>
<CR><LF>
*Status: 200 OK<CR><LF>
*Client-Side-Proxy-IP: 130.69.111.111<CR><LF>
*Client-Side-Proxy-Name:
University.of.Tokyo.Branch.Hospital<CR><LF>
*User-Agent-IP: 192.168.123.123<CR><LF>
*Server-Side-Proxy-IP: 130.69.222.222<CR><LF>
*Server-Side-Proxy-Name:
Coordinating.Center.CSCRG<CR><LF>
*Server-Side-Proxy-Port: 8080<CR><LF>
*Connection-ID: 6zdDfldfcZLj8V!i<CR><LF>
*Response-Nonce: ef23dc9b<CR><LF>
<CR><LF>
*03a606de822b52563a3171d8eb0fcf80
```